stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 204/82 AUGUSTUS

D.S.H. ROSENTHAL & P.J.W. TEN HAGEN

GKS IN C

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

CR Categories and Subject Descriptors:
I.3.4 [Computer Graphics]: Graphics Utilities — graphics packages;

General Terms: Standardization

GKS in C

by

David S. H. Rosenthal
Paul J. W. ten Hagen

ABSTRACT

A binding of GKS, the draft ISO standard for 2D graphics software, to the programming language C is presented, together with the principles underlying its design. The binding is compared with the binding to FORTRAN.

An implementation of GKS according to this binding is also described, particular attention being paid to the techniques for segment storage, input, and workstations. The implementation is assessed against implementations in FORTRAN.

This paper will be presented at the EuroGraphics '82 conference, in Manchester, on September 9thd 1982.

1. Introduction

"Can blue men sing the Whites,
or are they hypocrites
for singing Blue?"
Bonzo Dog Do-Dah Band

GKS[3], the ISO draft international standard for 2D graphics software, is specified in a language-independent form. It is anticipated that bindings of this standard to popular programming languages will be produced from various sources, though to date the only bindings available have been to FORTRAN, primarily from DIN[2].

A language binding for GKS specifies two things, how the abstract data types used in the GKS document are mapped onto data types supported by the host language, and how the abstract function names and argument lists of the document are mapped onto host language function specifications.

A language binding to the programming language C[6] has been developed, and is being used as the basis for an implementation of GKS for UNIX.* This binding appears to differ, in many respects, from the DIN FORTRAN binding. These differences stem from the principles used to map GKS concepts onto C language concepts, and provide an interesting sidelight on the GKS document.

2. The C Binding

The GKS document specifies rules for language bindings. As recently revised, they are:

1. All GKS functions, other than inquiry functions, must appear atomic to the application.
2. The language binding should specify, for each GKS abstract function name, exactly one identifier acceptable to the language.
3. The language binding should specify, for each of the GKS data types, a corresponding data type acceptable to the language. Other data types may be specified as convenient, in terms of the GKS data types.
4. The language binding should specify, for each GKS abstract function, how the corresponding language function is to be invoked, and the means whereby each of the input parameters is supplied to, and each of the output parameters received from, the language function.
5. The language binding should specify a set of identifiers, acceptable to the language, which may be used by an implementation for internal communication.

Using these rules, it would be simple to produce a binding that would work, by interpreting the GKS abstract function specifications directly as C function specifications. It would, however, force the application programmer to write a stilted, artificial dialect of C, and would impose considerable inefficiencies on the implementation. Two additional principles were therefore added.

a) Do not interpret the document so literally as to prevent the implementor or the application programmer making use of the full range of C's facilities.

b) Do not interpret the document so literally as to force inefficient techniques on the implementor.

Of course, if applied injudiciously these principles would vitiate the standard. A reason for publishing the C binding at this stage is to assist in the development of a body of "case law" to determine acceptable bindings.

The complete C binding is set out in the Appendix. The following sub-sections discuss those areas in which the additional principles found particular application.

2.1. Data Types

The C binding specifies each different GKS data item as an individual type, using the set of GKS abstract types. These are themselves defined using C's basic types. This has three advantages. The practical one is that this improves the capacity of *lint*[4], the C type-checking program, to find bugs in both the implementation and application programs.

* UNIX is a Trademark of Bell Laboratories.

The aesthetic one is that it encourages application programmers to specify meaningful types for values they supply to GKS. The implementation advantage is that the representations of types can be adjusted in the light of experience, without affecting the text of application programs.*

2.2. Control

GKS specifies that names for picture segments and workstations are supplied by the application. The specification for the CREATE SEGMENT operation is

```
        CREATE SEGMENT
        Parameter: I  segment name        N
```

with the name of the newly created segment an *input* parameter. Arbitrary names of this kind are essential for some languages, but they impose an extra burden on the graphics kernel. It must maintain an index of these application names, and search it on each segment operation to locate the appropriate segment information. The concept of the *open segment*, to which all output primitives are addressed may be partly traced to this point. Opening an individual segment avoids having to search the segment name index on every primitive.

C, unlike many languages, directly supports the concept of names for objects. An object, for example a segment, will be described by an instance of a (probably structured) data type. This instance will have an address, which is a value accessible to C, and forms the natural name for the object.

The natural definition for the CREATE SEGMENT operation in C would be as a function, returning the address of the instance of the structured data type (Seg) which described the newly created segment.

```
        Seg *newseg()
```

Thus the name of the newly created segment would be invented by the system.

However, this definition is inadequate. Segments may be written to a metafile, and then read back by a different program. Pointers as names are private to a particular program, and are not meaningful for a metafile used to communicate between programs. The correct definition for CREATE SEGMENT is

```
        Seg *newseg(n)
        Segname      n;
```

where **n** is the *external* name for the segment. Note the analogy with the way in which the UNIX **open** system call[7] provides a process with a local name for the global file object. In both cases, the external name is used only in the open operation, all other access is via the local name.

It might be thought that bundle indices were also candidates for implementation as pointers. However, the rôle of bundle indices is as names global to the set of workstations, both of this program and, via the metafile, to all GKS systems. These names are interpreted differently by each workstation; the contents of the bundle tables are workstation-specific.

2.3. Inquiries

A GKS application may find out system parameters and the current settings of state variables using inquiry functions. However, the C binding almost completely eliminates inquiry functions, as such. The application is provided with the data type definitions of the structures implementing the various GKS state lists. Whenever a state list is created, for example by OPEN SEGMENT, the corresponding function returns the address of the newly created state list structure.

The application could thus refer to the fields of the structure directly, but in doing so would insist that all state list structures were always in the program's address space. As, on smaller configurations, address space is scarce, this is not acceptable. It would seem attractive to replace the multitude of inquiry functions with a function for each state list whose arguments were the state list pointer and the

* *lint* does not enforce its rules, so application programmers are free to ignore its warnings. However, complaints from those who do so are not well received.

field name within it, but:

- field names alone are not valid expressions (and thus function arguments) in C
- specifying the type of the value returned by such a function is problematic.

Instead, the application invokes inquiry *macros*, looking like functions with these arguments. They are converted into in-line C by the pre-processor and thereby escape the restrictions on the use of field names. The definitions of the macros are part of the implementation, and provide hooks for paging the state lists between the program's address space and a file.

So that they may safely be called from error handling routines, GKS inquiry functions do not themselves generate errors. Instead, they set an output parameter indicating whether the value inquired for is available. In C, this precaution is not required. If the application has a valid state list address, then the value is available. Otherwise, the value is both unavailable and inaccessible.

The few remaining true inquiry functions are those computing a value from input parameters rather than returning a value from the state list. They are INQUIRE TEXT EXTENT, because it requires a string as input, and the three pixel inquiries

- INQUIRE PIXEL ARRAY DIMENSIONS
- INQUIRE PIXEL ARRAY
- INQUIRE PIXEL

both because they require rectangles or points as input, and because they should, if possible, be implemented using bit-map read-back from the device.

2.4. Attribute Settings

As the application has direct access to the state list structures, it is plausible to assume that the attribute setting functions might also be abolished, the application merely assigning values to the appropriate fields of the structures. Unfortunately, in some implementations some or all attribute settings may have side effects. For example, setting a segment's visibility attribute will cause visible changes to the display.

Further, GKS specifies that attribute values are checked for validity when they are set, not when they are used. Thus, simple assignments are inadequate; various operations may need to be invoked when the attribute is set.*

2.5. Errors

The normal C environment provides a file especially for error reporting (**stderr**). When a program is invoked, this file will normally be attached to the terminal, but may be directed elsewhere. The "error file" argument to OPEN GKS is thus superfluous, and indeed harmful, since if it is not set to **stderr** it will prevent the error file being re-directed.

The second, and more fundamental point is that it was decided to structure the binding so that as many errors as possible could be detected by *lint's* type-checking mechanism while the application was being constructed. Testing for these errors at run-time need not then be provided.

2.6. Input

The REQUEST <device class> functions of GKS suspend execution either until the operator supplies a value, or until the BREAK facility is invoked. The corresponding C functions return TRUE if a value was supplied, and FALSE if BREAK was invoked.

AWAIT EVENT is specified as a function returning a pointer to the current event. This is a structure containing the identification of the device generating the event, and a union of the types returned by each device class. GET <device class> functions are thus redundant, the application can refer to the appropriate fields of the current event structure directly.

* This does not prevent individual attribute setting functions being implemented as macros, if required.

2.7. Binding

Table 1 contains some statistics comparing the GKS document, the DIN FORTRAN binding, and the C binding. The C binding defines far fewer functions; on the other hand it defines many more data types.

Table 1 — Binding Statistics

	GKS	DIN	C
No. of "functions"	157	157	92
Max. no. arguments	13	28	6
Avg. no. arguments	3.85	4.98	2.32
No. of data types	8	8	62

As compared with the FORTRAN binding, the application programmer in C must be more concerned with the representation of information, both within the graphics system and the application. The correct structures must be used to hold information sent to and received from the system. On the other hand, the routine specifications are pleasantly terse.

Lint's automatic check that an application program adheres to the binding is an important feature missing from FORTRAN. The *lint* library, which is the basis for this test, was the form in which the binding was initially written. The inverse of this library, a dummy application program using all the routines defined by the binding, has proven valuable as an automatic check of the implementation.

3. Implementation

The C binding is being used as the basis for an implementation of GKS for UNIX. The implementation is proceeding in two stages, a "quick and dirty" system with no concern for efficiency but support for all GKS levels is being constructed, then those parts discovered to cause significant inefficiencies will be re-implemented. This approach is feasible because the implementation is initially based on Berkeley's virtual memory UNIX;[1] it is anticipated that the second phase will result in an implementation usable on all UNIXes.

3.1. Segment Storage

GKS requires two independent means for storing segments and the primitives they contain. Each workstation must be capable of storing segments sent to it, regenerating them and transforming them. If the hardware cannot support this, the segments must be stored and regenerated by software in the workstation driver. In addition, device independent segment storage (DISS) is required. This stores segments at an early stage in the transformation pipeline, so that they may subsequently be sent to workstations.

Although the two segment stores are logically distinct, it is possible to implement them using the same mechanism. The implementation does so; in the interest of simplicity both DISS and the software segment stores for workstations requiring them are implemented entirely in the heap.

3.2. Workstations

Given the complexity of real devices, the implementation of a large enough set of GKS workstations to be useful is also a problem. This exposes a dilemma. If the interface between the device independent and device dependent parts of GKS (the DI/DD interface) is pitched at a high level, then the implementation will be capable of exploiting the facilities of high-performance displays. On the other hand, the emulation of advanced facilities in each workstation makes writing workstation drivers difficult.

On the other hand, if the DI/DD interface is set at a low level, to make creating a wide range of drivers easy, then even intelligent devices will be treated as if they were stupid. There are two main reasons why the second course was nevertheless taken.

- Most UNIX installations are small and cannot afford high-performance devices. On the other hand, many already have low-performance devices.
- Existing graphics packages written in C[8] contain useful drivers for many low-cost devices. If the DI/DD interface is set sufficiently low, these can be re-used

almost intact.

During the second phase of the implementation we expect that the DI/DD interface will be moved up, enabling more facilities of high-performance devices to be exploited.

3.3. Input

GKS specifies two levels of input capability. At level 'b', only REQUEST mode input is provided, whereas at level 'c', both SAMPLE and EVENT modes must be provided. The underlying model of GKS input[9] describes the behaviour of input devices in terms of two autonomous processes for each device, a *measure* process determining its logical data value, and a *trigger* process determining the times at which events are recorded.

It is possible to implement REQUEST mode without using multiple processes or tasks. Since the GKS process is (conceptually) suspended during the time a device is satisfying a REQUEST, the process itself can simulate the device. Because the simulation of a logical device in SAMPLE or EVENT mode must proceed in parallel with the application, level 'c' input needs multiple processes or tasks.

Berkeley are at present improving and generalising the inter-process communication facilities of UNIX incorporating experience of connecting the system to various networks[5]. When these facilities become available, the GKS input model will be used directly as the basis for the level 'c' input facilities, with a separate UNIX process for each measure and trigger process. In the meantime, only REQUEST input is being provided.

3.4. Assessment

At the time of writing, the implementation has reached about level '1a', with some level '2a' routines also present. It is estimated that, including AED512 and Tektronix 4014 drivers, the level '2a' system will total less than 2000 lines of code.

4. Acknowledgements

During the development of the binding, David Rosenthal was supported by UK Science and Engineering Research Council grants GR/A80341 and N2B 1R 0371. The implementation is being funded by T. N. O., the Netherlands organisation for applied scientific research. It is largely the work of Behr de Ruiter.

REFERENCES

[1] Ö. Babaoğlu, W. N. Joy, and J. Porcar, "Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX Operating System," Computer Systems Research Group, Dept. EECS, University of California, Berkeley, California (December 1979).

[2] DIN, "FORTRAN Interface of GKS 7.0," Doc. 13-82, DIN NI-5.9, Darmstadt/Erlangen (March 1982).

[3] ISO, "Graphical Kernel System (GKS) — Functional Description," ISO DP 7942 (January 1982).

[4] S. C. Johnson, "Lint, a C Program Checker," Comp. Sci. Tech. Rep. No. 65, Bell Laboratories, Murray Hill, New Jersey (1978).

[5] W. N. Joy, E. Cooper, R. Fabry, S. Leffler, and K. McKusik, "4.2BSD System Manual," Computer Systems Research Group, Dept. EECS, University of California, Berkeley, California (February 1982).

[6] B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, Englewood Cliffs, New Jersey (1978).

[7] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Comm. Assoc. Comput. Mach.* **17**(7), pp.365-375 (July 1974).

[8] D. S. H. Rosenthal, "'Methodology in Computer Graphics' Re-examined," *Computer Graphics* **15**(2), pp.152-162 (July 1981).

[9] D. S. H. Rosenthal, J. C. Michener, G. Pfaff, R. Kessener, and M. Sabin, “The Detailed Semantics of Graphics Input Devices,” To be presented at, SIGGRAPH ’82, Boston, Mass (July 1982).

Appendix - The Complete C Binding

Warning

This binding is of GKS version 7.0. Since it was published, certain changes have been made to the proposal. They consist in the main of extra functions. Copy deadlines for this paper did not permit their incorporation. Anyone interested in using or commenting on this binding should contact the authors for an updated version.

Data Types

What follows is a sorted and abbreviated version of the “include” file supplied to application programs. It includes all the defined types, but excludes many of the fields of the state list structures. It was the second document created by the project.

```
typedef struct {
        Bindex  fa_ix;
        Istyle  fa_is;
        Sindex  fa_si;
        Cindex  fa_ci;
        } AreaRep;
typedef struct {
        Int     as_xsz;
        Int     as_ysz;
        } ArrSiz;

typedef Int Bindex;
typedef enum { FALSE, TRUE } Bool;

typedef Real Charef;
typedef Real Charht;
typedef Real Charsp;
typedef Int Choice;
typedef Int Cindex;
typedef struct {
        Real    c_red, c_green, c_blue;
        } Colour;
```

```
typedef struct {
        Int     d_x, d_y;
        } Dc;
typedef enum { ASAP,
        BNIL,
        BNIG,
        DALV } Defmode;
typedef Int Devno;
typedef union {
        struct {
                /* Impl. dependent */
                } loc_rec;
        struct {
                Realval_max, val_min;
                /* Impl. dependent */
                } val_rec;
        struct {
                /* Impl. dependent */
                } cho_rec;
        struct {
                /* Impl. dependent */
                } pik_rec;
        struct {
                Size str_size, str_posn;
                /* Impl. dependent */
                } str_rec;
        } Drecord;
typedef struct {
        Dc      d_ll, d_ur;
        } Drect;

typedef struct {
        /* Impl. dependent */
        } Erarea;
typedef Int Ercode;
typedef struct {
        /* Impl. dependent */
        } Es_data;
typedef enum {
        /* Impl. dependent */
        } Es_func;
typedef struct {
        Wss     *ev_ws;
        Devno   ev_devn;
        Iclass  ev_class;
        Idata   ev_data;
        } Event;

/* type File defined by "stdio" */
```

```
typedef enum {
        /* Impl. dependent */
        } Gdpi;
typedef struct {
        GksLevl gk_levl;
        Ntran   *gk_ntran[NO_NTRAN];
        Bindex  gk_line;
        Bindex  gk_mark;
        Bindex  gk_text;
        Bindex  gk_area;
        Wc      gk_patref;
        Wc      gk_patsiz;
        Charht  gk_chht;
        Charef  gk_chef;
        Wc      gk_chup;
        Path    gk_path;
        Charsp  gk_chsp;
        Wss     *gk_opws[NO_WSS];
        Wss     *gk_actv[NO_WSS];
        Seg     *gk_opsg;
        Seginst *gk_segs;
        Pickid  gk_pikid;
        Bool    gk_clip;
        Bool    gk_more;
        } Gks;
typedef enum { GCKL,
        GKOP,
        WSOP,
        WSAC,
        SGOP } GksState;
typedef struct {
        char    op_lev, ip_lev;
        } GksLevl;
typedef char    *Grecord;

typedef enum { LOCATOR,
        VALUATOR,
        CHOICE,
        PICK,
        STRING } Iclass;
typedef union {
        Loc     ev_loc;
        Val     ev_val;
        Choice  ev_cho;
        Pick    ev_pik;
        String  ev_str;
        } Idata;
typedef struct {
        Iclass  id_clas;
        Imode   id_mode;
        Bool    id_echo;
        Drect   id_area;
        Pet     id_pet;
        Idata   id_ival;
        Drecord id_drec;
        } Idevice;
typedef enum { REQUEST,
        EVENT,
        SAMPLE } Imode;
typedef unsigned Int;
typedef enum { HOLLOW,
        SOLID,
        PATTERN,
        HATCH } Istyle;

typedef struct {
        Bindex  lr_ix;
        Ltype   lr_lt;
        Lwidth  lr_lw;
        Cindex  lr_ci;
        } LineRep
typedef struct {
        Ntran   *loc_nt;
        Wc      loc_pt;
        } Locate;
typedef Int Ltype;
typedef Real Lwidth;
typedef struct {
        Bindex  mr_ix;
        Mtype   mr_mt;
        Msize   mr_ms;
        Cindex  mr_ci;
        } MarkRep;
typedef Real Msize;
typedef Int Mtype;

typedef  struct {
        Real    n_x, n_y;
        } Nc;
typedef struct {
        Nc      n_ll, n_ur;
        } Nrect;
typedef struct {
        Wrect   nt_wind;
        Nrect   nt_view;
        } Ntran;

typedef enum { RIGHT,
        LEFT,
        UP,
        DOWN } Path;
typedef struct {
        Bindex  pa_ix;
        Int     pa_n, pa_m;
        Cindex  pa_ci;
        } PattRep;
typedef Int Pet;
typedef struct {
        Seg     *pik_seg;
        Pickid  pik_pid;
        } Pick;
typedef Int Pickid;

typedef float Real;
```

```
typedef struct {
        Segname se_name;
        Bool    se_visb;
        Bool    se_detc;
        Bool    se_hilt;
        Segpri  se_prio;
        Wss     *se_onws[NO_WSS];
        Tmat    se_tran;
        } Seg;
typedef struct SEGINST {
        struct SEGINST  *si_next;
        struct SEGINST  *si_prev;
        Seg             *si_this;
        } Seginst;
typedef Int Segname;
typedef Real Segpri;
typedef Int Sindex;
typedef Int Size
typedef char *String;

typedef struct {
        Bindex  tx_ix;
        Tfont   tx_tf;
        Tprec   tx_tp;
        Cindex  tx_ci;
        } TextRep;
typedef Int Tfont;
typedef Real Tmat[2,3];
typedef enum { STRING,
        CHAR,
        STROKE } Tprec;

typedef Real Value;

typedef struct {
        Real    w_x, w_y;
        } Wc;
typedef struct {
        Wc      w_ll, w_ur;
        } Wrect;

typedef struct {
        /* Too complex for inclusion */
        } Wsd;
```

```
typedef struct {
        Wsd     *ws_wsd;
        Bool    ws_actv;
        Defmodews_defr;
        Bool    ws_imprg;
        Bool    ws_waitr;
        Nrect   ws_reqw;
        Nrect   ws_curw;
        Drect   ws_reqv;
        Drect   ws_curv;
        Int     ws_nline;
        LineRep *ws_line;
        Int     ws_nmark;
        MarkRep     *ws_mark;
        Int     ws_ntext;
        TextRep *ws_text;
        Int     ws_narea;
        AreaRep *ws_area;
        Int     ws_ncolr;
        Colour  *ws_colr;
        Seginst *ws_segs;
        Idevice *ws_devs;
        } Wss;
```

Routines

What follows is a slightly abbreviated version of the *lint* library file, which is used to test application programs for conformance to the language binding. All function definitions are included, but additional information used only by *lint* is excluded. It was the first document created by the project.

```
extern GksState   opstate;
/* OPEN GKS */
Gks *
open_gks(erh,era,s)
Ercode  (*erh)();
Erarea   *era;
Size      s;

/* CLOSE GKS */
close_gks()

/* OPEN W.S. */
Wss *
open_ws(dev,wsd)
File      *dev;
Wsd       *wsd;

/* CLOSE W.S. */
close_ws(ws)
Wss       *ws;

/* ACTIVATE W.S. */
activate(ws)
Wss       *ws;

/* DEACTIVATE W.S. */
deactivate(ws)
Wss       *ws;

/* CLEAR W.S. */
clear(ws)
Wss       *ws;

#if (OP_LEVEL > 0)
/* REDRAW ALL SEGMENTS ON W.S. */
redraw(ws)
Wss       *ws;
#endif (OP_LEVEL > 0)

/* UPDATE */
update(ws)
Wss       *ws;

/* MESSAGE */
message(ws,s)
Wss       *ws;
String    *s;

/* ESCAPE */
escape(ef,ep)
Es_func ef;
Es_data *ep;

/* SET WINDOW */
s_window(nt,wr)
Ntran    *nt;
Wrect    *wr;

/* SET VIEWPORT */
s_viewport(nt,nr)
Ntran    *nt;
Nrect    *nr;
```

```
#if (IP_LEVEL > 0)
/* SET VIP */
s_vip(nt,rnt,hi)
Ntran    *nt, *rnt;
Bool     hi;
#endif (IP_LEVEL > 0)

/* SELECT NORMALIZATION TRAN */
s_cntran(nt)
Ntran    *nt;

/* SET CLIPPING INDICATOR */
s_clip(flg)
Bool     flg;

/* SET W.S. WINDOW */
s_w_wind(ws,nr)
Wss      *ws;
Nrect    *nr;

/* SET W.S. VIEWPORT */
s_w_vprt(ws,dr)
Wss      *ws;
Drect    *dr;

/* SET CHARACTER HEIGHT */
s_ch_ht(ht)
Charht   ht;

#if (OP_LEVEL > 0)
/* SET CHAR. EXP. FACTOR */
s_ch_ef(f)
Charef   f;

/* SET CHAR. UP VECTOR */
s_ch_up(v)
Wc       *v;

/* SET CHARACTER PATH */
s_ch_pt(pt)
Path     pt;
#endif (OP_LEVEL > 0)

/* SET CHARACTER SPACING */
s_ch_sp(sp)
Charsp   sp;

/* SET POLYLINE INDEX */
s_pl_i(i)
Bindex   i;

/* SET POLYMARKER INDEX */
s_pm_i(i)
Bindex   i;

/* SET TEXT INDEX */
s_tx_i(i)
Bindex   i;

/* SET FILL AREA INDEX */
s_fa_i(i)
Bindex   i;

#if (OP_LEVEL > 0)
/* SET PATTERN REF. POINT */
s_patpt(rp)
Wc       *rp;

/* SET PATTERN SIZE */
s_patsiz(ps)
Wc       *ps;

#if (IP_LEVEL > 0)
/* SET PICK ID */
s_pikid(id)
Pickid   id;
#endif (IP_LEVEL > 0)

/* SET VISIBILITY */
s_segvis(seg,vis)
Seg      *seg;
Bool     flg;

#if (IP_LEVEL > 0)
/* SET DETECTABILITY */
s_detect(seg,det)
Seg      *seg;
Bool     det;
#endif (IP_LEVEL > 0)

/* SET HIGHLIGHTING */
s_hilite(seg,hlt)
Seg      *seg;
Bool     hlt;
```

```
/* SET SEGMENT PRIORITY */
s_segpri(seg,pri)
Seg       *seg;
Segpri    pri;

/* SET POLYLINE REPR. */
pl_rep(ws,pl)
Wss       *ws;
LineRep *pl;

/* SET POLYMARKER REPR. */
pm_rep(ws,mk)
Wss       *ws;
MarkRep            *mk;

/* SET TEXT REPR. */
tx_rep(ws,tx)
Wss       *ws;
TextRep *tx;

/* SET FILL AREA REPR. */
fa_rep(ws,fa)
Wss       *ws;
AreaRep *fa;

/* SET PATTERN REPR. */
pat_rep(ws,pa)
Wss       *ws;
PattRep *pa;
#endif (OP_LEVEL > 0)

/* SET COLOUR REPR. */
col_rep(ws,ci,col)
Wss       *ws;
Bindex   ci;
Colour   *col;

#if (OP_LEVEL > 0)
/* SET DEFERRAL STATE */
s_defer(ws,def,reg)
Wss       *ws;
Defmodedef;
Bool      reg;
#endif (OP_LEVEL > 0)
```

```
/* POLYLINE */
polyline(n,p)
Size      n;
Wc        *p;

/* POLYMARKER */
polymark(n,p)
Size      n;
Wc        *p;

/* TEXT */
text(p,s)
Wc        *p;
String    s;

/* FILL AREA */
fillarea(n,p)
Size      n;
Wc        *p;

/* PIXEL ARRAY */
pixels(wr,n,m,ci)
Wrect     *wr;
Size      n, m;
Cindex    *ci;

/* GENERALIZED DRAWING PRIM. */
g_draw(n,p,id,m,dr)
Size      n;
Wc        *p;
Gdpi      id;
Size      m;
Grecord *dr;

#if (OP_LEVEL > 0)
/* CREATE SEGMENT */
Seg *
newseg(n)
Segname n;

/* CLOSE SEGMENT */
closeg()

/* DELETE SEGMENT */
zapseg(seg)
Seg       *seg;
```

```
/* DELETE SEG. FROM W.S. */
delseg(ws,seg)
Wss      *ws;
Seg      *seg;

/* TRANSFORM SEGMENT */
transeg(seg,mat)
Seg      *seg;
Tmat     *mat;

#if (OP_LEVEL > 1)
/* ASSOCIATE SEG. WITH W.S. */
sendseg(ws,seg)
Wss      *ws;
Seg      *seg;

/* COPY SEG. TO W.S. */
copyseg(ws,seg)
Wss      *ws;
Seg      *seg;

/* INSERT SEGMENT */
insseg(seg,mat)
Seg      *seg;
Tmat     *mat;
#endif (OP_LEVEL > 1)
#endif (OP_LEVEL > 0)

#if (IP_LEVEL > 0)
/* INITIALIZE LOCATOR */
init_loc(ws,dn,val,pet,ea,dr)
Wss      *ws;
Devno    dn;
Locate   *val;
Pet      pet;
Drect    *ea;
Drecord *dr;

/* INITIALIZE VALUATOR */
init_val(ws,dn,val,pet,ea,dr)
Wss      *ws;
Devno    dn;
Value    val;
Pet      pet;
Drect    *ea;
Drecord *dr;

/* INITIALIZE CHOICE */
init_cho(ws,dn,val,pet,ea,dr)
Wss      *ws;
Devno    dn;
Choice   *val;
Pet      pet;
Drect    *ea;
Drecord *dr;

if (OP_LEVEL > 1)
/* INITIALIZE PICK */
init_pik(ws,dn,val,pet,ea,dr)
Wss      *ws;
Devno    dn;
Pick     *val;
Pet      pet;
Drect    *ea;
Drecord *dr;
#endif (OP_LEVEL > 1)

/* INITIALIZE STRING */
init_str(ws,dn,val,pet,ea,dr)
Wss      *ws;
Devno    dn;
String   val;
Pet      pet;
Drect    *ea;
Drecord *dr;

/* SET LOCATOR MODE */
loc_mode(ws,dn,mo,echo)
Wss      *ws;
Devno    dn;
Imode    mo;
Bool     echo;

/* SET VALUATOR MODE */
val_mode(ws,dn,mo,echo)
Wss      *ws;
Devno    dn;
Imode    mo;
Bool     echo;
```

```
/* SET CHOICE MODE */
cho_mode(ws,dn,mo,echo)
Wss      *ws;
Devno    dn;
Imode    mo;
Bool     echo;

#if (OP_LEVEL > 1)
/* SET PICK MODE */
pik_mode(ws,dn,mo,echo)
Wss      *ws;
Devno    dn;
Imode    mo;
Bool     echo;
#endif (OP_LEVEL > 1)

/* SET STRING MODE */
str_mode(ws,dn,mo,echo)
Wss      *ws;
Devno    dn;
Imode    mo;
Bool     echo;

/* REQUEST LOCATOR */
Bool
req_loc(ws,dn,val)
Wss      *ws;
Devno    dn;
Locate   *val;

/* REQUEST VALUATOR */
Bool
req_val(ws,dn,val)
Wss      *ws;
Devno    dn;
Value    *val;

/* REQUEST CHOICE */
Bool
req_cho(ws,dn,val)
Wss      *ws;
Devno    dn;
Choice   *val;

#if (OP_LEVEL > 1)
/* REQUEST PICK */
Bool
req_pik(ws,dn,val)
Wss      *ws;
Devno    dn;
Pick     *val;
#endif (OP_LEVEL > 1)

/* REQUEST STRING */
Bool
req_str(ws,dn,val)
Wss      *ws;
Devno    dn;
String   val;

#if (IP_LEVEL > 1)
/* AWAIT EVENT */
Event *
await(sec)
Real     sec;

/* FLUSH DEVICE EVENTS */
flsh_ev(ws,class,dn)
Wss      *ws;
Iclass   class;
Devno    dn;

/* N.B. no GET <class> Functions! */

/* SAMPLE LOCATOR */
smp_loc(ws,dn,val)
Wss      *ws;
Devno    dn;
Locate   *val;

/* SAMPLE VALUATOR */
smp_val(ws,dn,val)
Wss      *ws;
Devno    dn;
Value    *val;

/* SAMPLE CHOICE */
smp_cho(ws,dn,val)
Wss      *ws;
Devno    dn;
Choice   *val;
```

```
/* SAMPLE PICK */
smp_pik(ws,dn,val)
Wss      *ws;
Devno    dn;
Pick     *val;

/* SAMPLE STRING */
smp_str(ws,dn,val)
Wss      *ws;
Devno    dn;
String   val;

#endif (IP_LEVEL > 1)
#endif (IP_LEVEL > 0)
/* N.B. metafile functions omitted */
         WRITE_ITEM
         GET ITEM TYPE
         READ ITEM
         INTERPRET

/*
 * These aren't functions but macros.
 * They return a value of the
 * type of the argument 'fn',
 * which is a field name of the
 * structure involved.
 *
/* INQUIRE GKS STATE */
inq_gks(gk,fn)
Gks      *gk;
/* INQUIRE W.S. DESCRIPTION */
inq_wsd(ws,fn)
Wsd      *ws;
/* INQUIRE W.S. STATE */
inq_wss(ws,fn)
Wsd      *ws;
/* INQUIRE SEGMENT STATE */
inq_seg(sg,fn)
Seg      *sg;
 *
 */

Bool
pxl_siz(ws,r,sz)
Wss      *ws;
Wrect    *r;
ArrSiz   *sz;

Bool
pxl_arr(ws,r,sz,inv,arr)
Wss      *ws;
Wrect    *r;
ArrSiz   *sz;
Bool     *inv;
Cindex   *arr;

Bool
pixel(ws,p,ci)
Wss      *ws;
Wc       *p;
Cindex   ci;

/* INQUIRE TEXT EXTENT */
Bool
textext(ws,pt,st,cat,tr)
Wss      *ws;
Wc       *pt;
String   st;
Wc       *cat;
Wrect    *tr;

/* N.B. utility functions omitted */
         EMERGENCY CLOSE GKS
         ACCUMULATE TRANSFORMATION MATRIX
         SET TRANSFORMATION MATRIX
```

MC NR

35226